DSCI471 Final Project

Image Caption Generator

Kevin Shi, Rithvik Sukumaran

Proposal

- Generate captions for an image
- Use traditional means as well as more novel techniques
  - CNN + LSTM
  - Transformers, both visual and textual
- Flickr30k dataset for caption training
  - https://huggingface.co/datasets/nlphuji/flickr30k
  - 30k images with 5 captions per image, making 150k total observations
  - Had to scale back to Flickr8k, which has 6k images in the train/test split
- Cifar10 for classifier training
  - 32x32 images, 10 classes, 50k images in training split, evenly split among the classes

Image Caption Generator

- Annotate an image with relevant captions by recognizing its contents using deep learning and CV
- How?
  - Label an image with words with the help of flickr datasets
  - The imagenet dataset trains the CNN model called Xception
  - Xception is responsible for image feature extraction
  - LSTM model uses the feature extraction to generate the image caption

Image Caption Generator Model (CNN-RNN model) = CNN + LSTM

- CNN – To extract features from the image. A pre-trained model called Xception is used for this.
- LSTM – To generate a description from the extracted information of the image.

Source: https://www.analyticsvidhya.com/blog/2021/12/step-by-step-guide-to-build-image-caption-generator-using-deep-learning/

Project Disclaimer

1. We followed the tutorials from data-flair.training and analyticsvidhya.com for this project.
2. Our original plan was to train an image caption generator on the flickr30k dataset. However, we ran into multiple issues:
    a. Long training time (22 hours to train)
    b. Setting up and running in tux (storage, network interruptions, etc.)
    c. Saving the model (.keras extension works, but .h5 does not work)
3. Since the flickr30k dataset took too long to train, we scaled back to using the flickr8k dataset. We decided to first get a working model with the flickr8k dataset and then scale back up to the flickr30k dataset if we had time.

Step 1: Explore the Dataset

Flickr8k Dataset - https://huggingface.co/datasets/jxie/flickr8k

Explore Dataset

```
[ ] type(dataset)
```

```
datasets.dataset_dict.IterableDatasetDict
```

```
[ ] print(dataset)
```

```
IterableDatasetDict({
    train: IterableDataset({
        features: ['image', 'caption_0', 'caption_1', 'caption_2', 'caption_3', 'caption_4'],
        n_shards: 2
    })
    validation: IterableDataset({
        features: ['image', 'caption_0', 'caption_1', 'caption_2', 'caption_3', 'caption_4'],
        n_shards: 1
    })
    test: IterableDataset({
        features: ['image', 'caption_0', 'caption_1', 'caption_2', 'caption_3', 'caption_4'],
        n_shards: 1
    })
})
```

Step 2: Perform Data Cleaning

- Create a description dictionary that will map images with all 5 captions
- Convert all upper case alphabets to lowercase, removing punctuations and words containing numbers
- Create a vocabulary

```
print(len(cleaned_descriptions))
print(type(cleaned_descriptions))
print(next(iter(cleaned_descriptions.items())))
```

```
6000
<class 'dict'>
('image_0', ['a black dog is running after a white dog in the snow', 'black dog chasing brown dog through snow', 'two dogs chase each other across the snowy
```

Step 3: Extract Feature Vector Using Xception

Extracting the feature vector (do beforehand)

```
'''
# Feature extraction
def preprocess_image(image, target_size=(299, 299)):
    image = image.resize(target_size)
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0)
    image = preprocess_input(image)
    return image

model = Xception(include_top=False, pooling='avg')

def extract_features(image, model):
    preprocessed_image = preprocess_image(image)
    features = model.predict(preprocessed_image)
    return features

features_dict = {}
for idx, example in tqdm(enumerate(train_dataset)):
    image = example['image']
    features = extract_features(image, model)
    key = f'image_{idx}'
    features_dict[key] = features
'''
```

The cell above took around 1 hour 10 mins to run

```
[ ] '''
    import pickle

    # Save features to a file
    with open('/content/drive/My Drive/Colab Notebooks/dsci471/features_dict_8k_2.pkl', 'wb') as f:
        pickle.dump(features_dict, f)
    '''
```

Step 4: Loading dataset for Training the model

- Shuffle and split the dataset
- Define the function to load cleaned descriptions with <start> and <end> tokens
- Load features for training images
- Tokenize the vocabulary

Tokenizing the Vocabulary

Machines are not familiar with complex English words so, to process model'sdata they need a simple numerical representation. That's why we map every word of the vocabulary with a separate unique index value. An in-built tokenizer function is present in the Keras library to create tokens from our vocabulary. We can save them to a pickle file named "tokenizer.p".

Step 5: Create a Data generator

Create Data generator

```
[ ]  def data_generator(captions, features, tokenizer, max_length):
         while True:
             for key, captions_list in captions.items():
                 feature = features[key][0]
                 inp_image, inp_seq, op_word = create_sequences(tokenizer, max_length, captions_list, feature)
                 yield (inp_image, inp_seq), op_word

     def create_sequences(tokenizer, max_length, captions_list, feature):
         x_1, x_2, y = list(), list(), list()
         for caption in captions_list:
             seq = tokenizer.texts_to_sequences([caption])[0]
             for i in range(1, len(seq)):
                 in_seq, out_seq = seq[:i], seq[i]
                 in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                 out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                 x_1.append(feature)
                 x_2.append(in_seq)
                 y.append(out_seq)
         return np.array(x_1), np.array(x_2), np.array(y)
```

```
# Check the shape of the input and output for the model
[a,b],c = next(data_generator(train_descriptions, train_features, tokenizer, max_length))
a.shape, b.shape, c.shape
```

```
((50, 2048), (50, 38), (50, 6883))
```

Step 6: Define the CNN-RNN model

Defining the CNN-RNN model

```
# Define the model
from keras.layers import add
from keras.utils import plot_model

def define_model(vocab_size, max_length):
    inputs1 = Input(shape=(2048,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)

    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)

    decoder1 = add([fe2, se3])
    decoder2 = Dense(256, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)

    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')
    print(model.summary())
    plot_model(model, to_file='model.png', show_shapes=True)
    return model
```

Step 7: Training the Image Caption Generator model

```python
model = define_model(vocab_size, max_length)

epochs = 10
steps = len(train_descriptions)

for i in range(epochs):
    generator = data_generator(train_descriptions, train_features, tokenizer, max_length)
    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)
    if i == 0 or i == 9:
      model.save(f'/content/drive/My Drive/Colab Notebooks/dsci471/models3/model_8k_{i}.keras')
```

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 38)	0	–
input_layer (InputLayer)	(None, 2048)	0	–
embedding (Embedding)	(None, 38, 256)	1,762,048	input_layer_1[0][0]
dropout (Dropout)	(None, 2048)	0	input_layer[0][0]
dropout_1 (Dropout)	(None, 38, 256)	0	embedding[0][0]
not_equal (NotEqual)	(None, 38)	0	input_layer_1[0][0]
dense (Dense)	(None, 256)	524,544	dropout[0][0]
lstm (LSTM)	(None, 256)	525,312	dropout_1[0][0], not_equal[0][0]
add (Add)	(None, 256)	0	dense[0][0], lstm[0][0]
dense_1 (Dense)	(None, 256)	65,792	add[0][0]
dense_2 (Dense)	(None, 6883)	1,768,931	dense_1[0][0]

Total params: 4,646,627 (17.73 MB)
Trainable params: 4,646,627 (17.73 MB)
Non-trainable params: 0 (0.00 B)

None

```
4800/4800 ━━━━━━━━━━━━━━━━━━━━ 1644s 342ms/step - loss: 4.5669
4800/4800 ━━━━━━━━━━━━━━━━━━━━ 1575s 328ms/step - loss: 3.3291
4800/4800 ━━━━━━━━━━━━━━━━━━━━ 1599s 333ms/step - loss: 2.9938
4800/4800 ━━━━━━━━━━━━━━━━━━━━ 1544s 322ms/step - loss: 2.7936
4800/4800 ━━━━━━━━━━━━━━━━━━━━ 1670s 348ms/step - loss: 2.6579
4800/4800 ━━━━━━━━━━━━━━━━━━━━ 1573s 328ms/step - loss: 2.5582
4800/4800 ━━━━━━━━━━━━━━━━━━━━ 1571s 327ms/step - loss: 2.4722
4800/4800 ━━━━━━━━━━━━━━━━━━━━ 1609s 335ms/step - loss: 2.4098
4800/4800 ━━━━━━━━━━━━━━━━━━━━ 1621s 338ms/step - loss: 2.3577
4800/4800 ━━━━━━━━━━━━━━━━━━━━ 1595s 332ms/step - loss: 2.3134
```

Took around 4.44 hours to train

Step 8: Testing the Model

1

```
from keras.models import load_model

# Load the Keras model
model = load_model('/content/drive/MyDrive/Colab Notebooks/dsci471/models3/model_8k_9.keras')
```

2

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import load_model
from tensorflow.keras.applications.xception import Xception
import pickle
from google.colab import files
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer

# Upload the image
uploaded = files.upload()

# Set the image path
img_path = next(iter(uploaded))  # This will get the name of the uploaded image
```

```
Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Screen Shot 2024-06-04 at 10.58.52 PM.png to Screen Shot 2024-06-04 at 10.58.52 PM (7).png
```

3

```
def extract_features(filename, model):
    try:
        image = Image.open(filename)
    except:
        print("ERROR: Couldn't open image! Make sure the image path and extension is correct")
    image = image.resize((299,299))
    image = np.array(image)
    # for images that has 4 channels, we convert them into 3 channels
    if image.shape[2] == 4:
        image = image[..., :3]
    image = np.expand_dims(image, axis=0)
    image = image/127.5
    image = image - 1.0
    feature = model.predict(image)
    return feature

def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
     if index == integer:
         return word
    return None

def generate_desc(model, tokenizer, photo, max_length):
    in_text = 'start'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        pred = model.predict([photo,sequence], verbose=0)
        pred = np.argmax(pred)
        word = word_for_id(pred, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'end':
            break
    return in_text

max_length = 38

# Load the tokenizer
with open("/content/drive/My Drive/Colab Notebooks/dsci471/tokenizer3.pkl", "rb") as handle:
    tokenizer = pickle.load(handle)
```

4

```
# Load the Xception model
xception_model = Xception(include_top=False, pooling="avg")

# Extract features from the image
photo = extract_features(img_path, xception_model)
img = Image.open(img_path)
description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
```

```
start a soccer game end
<matplotlib.image.AxesImage at 0x7ef75861a2f0>
```

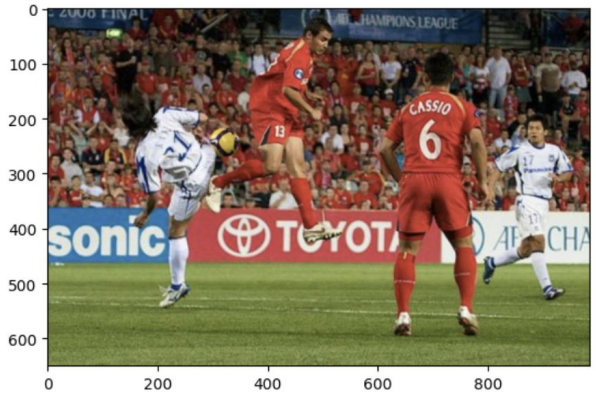
Actual Captions:

A group of men play soccer on the field .	A man in a red uniform leaps in the air as one in a white unifor…	During a soccer game , one man is in the air while another has one…	The soccer player in white is challenging the player in red fo…	Two soccer teams are playing , one is in white the other in red…

Good Results

```
# Load the Xception model
xception_model = Xception(include_top=False, pooling="avg")

# Extract features from the image
photo = extract_features(img_path, xception_model)
img = Image.open(img_path)
description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim
83683744/83683744 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step


start a soccer game end
<matplotlib.image.AxesImage at 0x7e7990361660>
```



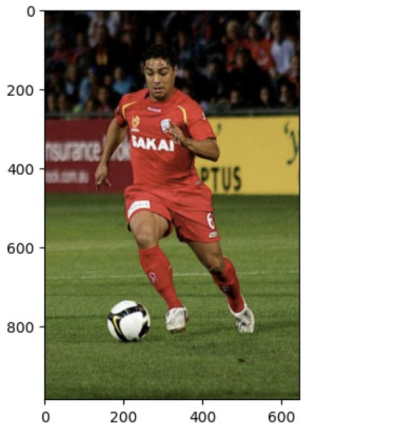
Actual Captions:

A man in a red soccer uniform getting ready to kick the ball .	A man in a red uniform plays in a soccer game .	A man in a red uniform runs towards a soccer ball on a field…	A player in a red uniform gets ready to kick the ball during a…	A soccer player in red uniform runs after a soccer ball .

Good Results

```
# Load the Xception model
xception_model = Xception(include_top=False, pooling="avg")

# Extract features from the image
photo = extract_features(img_path, xception_model)
img = Image.open(img_path)
description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step


start a soccer game in progress end
<matplotlib.image.AxesImage at 0x7e799079d750>
```



Actual Captions:

A brown dog in the snow has something hot pink in its mouth .	A brown dog in the snow holding a pink hat .	A brown dog is holding a pink shirt in the snow .	A dog is carrying something pink in its mouth while walking…	A dog with something pink in its mouth is looking forward .

Good Results

```
# Load the Xception model
xception_model = Xception(include_top=False, pooling="avg")

# Extract features from the image
photo = extract_features(img_path, xception_model)
img = Image.open(img_path)
description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step


start a dog is running through the snow end
<matplotlib.image.AxesImage at 0x7e79903ef730>
```

Actual Captions:

A car sponsored by Riwal is smoking its tires on a wet road .	A race car drives along a track in the rain	A racing car is driving around a wet track with its headlights on…	A small race car with advertising is driving on a rainy track .	A sponsored race car is driving with lights on a wet raceway .

Good Results

```
# Load the Xception model
xception_model = Xception(include_top=False, pooling="avg")

# Extract features from the image
photo = extract_features(img_path, xception_model)
img = Image.open(img_path)
description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step


start a race car is driving a corner during a track end
<matplotlib.image.AxesImage at 0x7e7990505450>
```

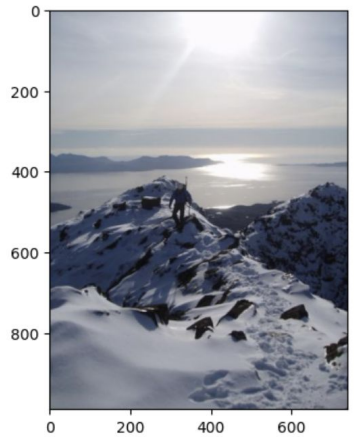
Actual Captions:

A climber walks along a snowy peak .	A man climbs over snow covered rocks by a shore .	A man is standing on a snowcapped mountain .	A person hikes down a snowy mountain .	There man on top of an snow covered mountain with a lake…

Good Results

```
# Load the Xception model
xception_model = Xception(include_top=False, pooling="avg")

# Extract features from the image
photo = extract_features(img_path, xception_model)
img = Image.open(img_path)
description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step


start a man in a red shirt and backpack is standing on a snowy mountain end
<matplotlib.image.AxesImage at 0x7e7990b25fc0>
```



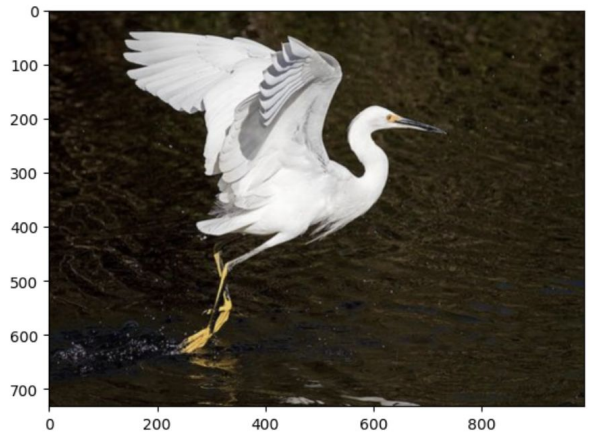
Actual Captions:

A large , white bird skates across the top of some water .	A white bird with a long neck flying low over the water	The large white bird grazes the water .	White bird in water preparing to fly .	White bird starting to take flight from a lake .

Good Results

```
# Load the Xception model
xception_model = Xception(include_top=False, pooling="avg")

# Extract features from the image
photo = extract_features(img_path, xception_model)
img = Image.open(img_path)
description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step


start a white bird is flying over the water end
<matplotlib.image.AxesImage at 0x7e7990817df0>
```



Actual Captions:

A grey bird stands majestically on a beach while waves roll in .	A large bird stands in the water on the beach .	A tall bird is standing on the sand beside the ocean .	A water bird standing at the ocean 's edge .	A white crane stands tall as it looks out upon the ocean .

Mediocre Results

```
# Load the Xception model
xception_model = Xception(include_top=False, pooling="avg")

# Extract features from the image
photo = extract_features(img_path, xception_model)
img = Image.open(img_path)
description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
```

```
start a white bird is flying through the air end
<matplotlib.image.AxesImage at 0x7e7983aee6e0>
```

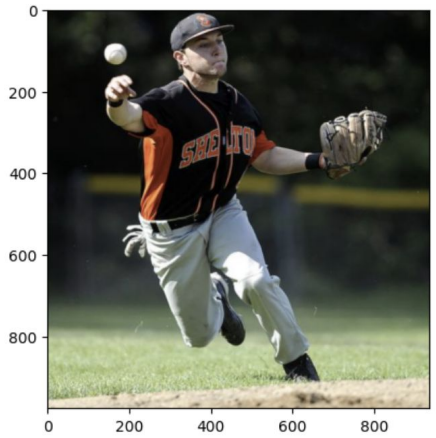
Actual Captions:

A baseball pitcher with Shelton on his shirt throws the ball .	A baseball player in a black and orange shirt , and with a glove…	A male dressed in a sports outfit tries to catch a baseball .	A man playing baseball .	The baseball player is running after the ball .

Mediocre Results

```
# Load the Xception model
xception_model = Xception(include_top=False, pooling="avg")

# Extract features from the image
photo = extract_features(img_path, xception_model)
img = Image.open(img_path)
description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step


start a baseball player in a white uniform is hitting the ball end
<matplotlib.image.AxesImage at 0x7e7983f5d5d0>
```



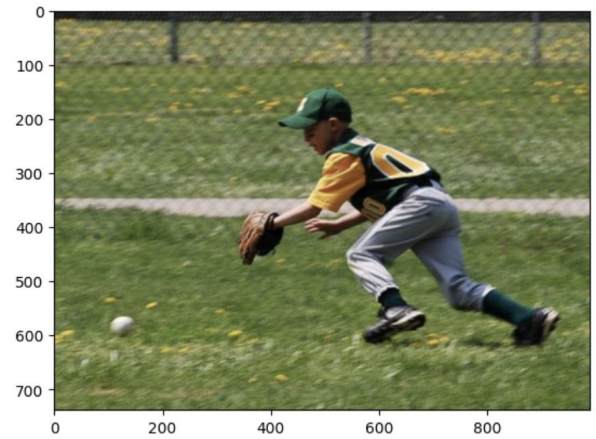
Actual Captions:

A boy baseball player in a green and yellow jersey leaps for the…	A boy chasing a baseball .	A little boy playing baseball goes for the ball on the ground .	A young boy dives to catch the ball during a baseball game .	The boy in the green and yellow shirt has a glove on and is goin…

Mediocre Results

```
# Load the Xception model
xception_model = Xception(include_top=False, pooling="avg")

# Extract features from the image
photo = extract_features(img_path, xception_model)
img = Image.open(img_path)
description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step


start a boy in a white uniform is playing baseball end
<matplotlib.image.AxesImage at 0x7e79903baf50>
```



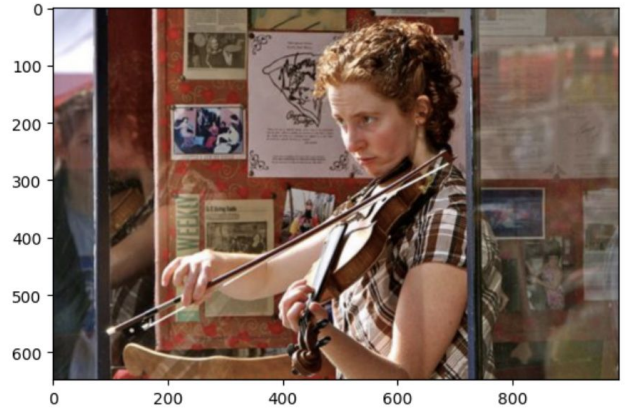
Actual Captions:

A curly haired woman plays the violin .	A red haired woman is holding a violin and boe .	A woman is playing a violin while surrounded by transparent walls…	A woman playing the violin with posters in the background .	The woman is holding a violin .

Bad Results

```
# Load the Xception model
xception_model = Xception(include_top=False, pooling="avg")

# Extract features from the image
photo = extract_features(img_path, xception_model)
img = Image.open(img_path)
description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step


start a man in a red shirt and sunglasses is sitting on a bench end
<matplotlib.image.AxesImage at 0x7e79906f4130>
```



Possible limitations:

- Too much contrast
- Too many colors
- Not enough similar image records

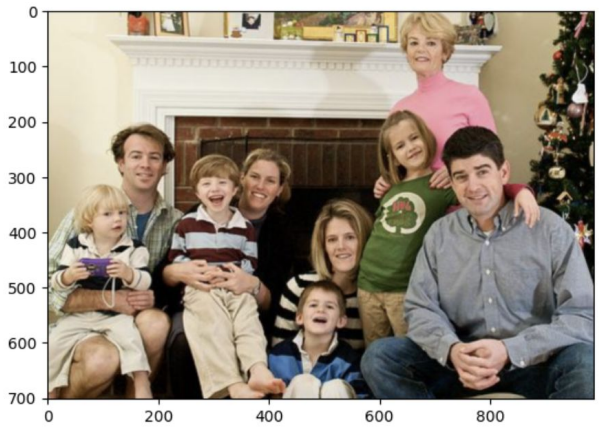
Actual Captions:

A family of nine people , including four children , pose i…	A family poses in front of the fireplace and Christmas tree .	A family posing by the mantle and christmas tree .	A happy family poses by the fireplace .	Two couples and four kids pose for a family picture .

Bad Results

```
# Load the Xception model
xception_model = Xception(include_top=False, pooling="avg")

# Extract features from the image
photo = extract_features(img_path, xception_model)
img = Image.open(img_path)
description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step


start a woman in a red shirt is sitting on a park bench end
<matplotlib.image.AxesImage at 0x7e798385dff0>
```



Possible limitations:

- Too much contrast
- Too many colors
- Too many figures/people
- Not enough similar image records

Actual Captions:

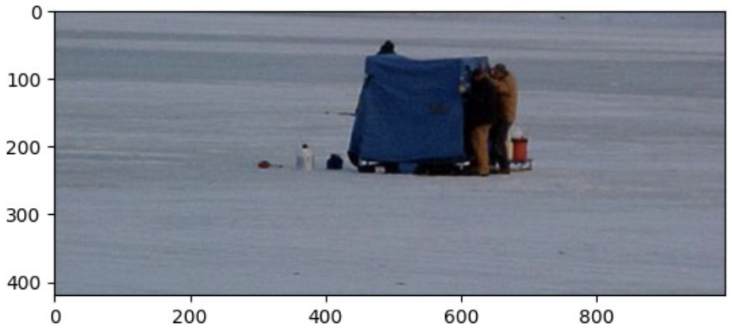
A tent is being set up on the ice .	Two men are about to enter an ice fishing tent on a snow covered…	two men setting up a blue ice fishing hut on an iced over lake	Two men , standing on an ice , looking into something covered…	Two people standing outside a blue tent structure on a snowy…

Bad Results

```
# Load the Xception model
xception_model = Xception(include_top=False, pooling="avg")

# Extract features from the image
photo = extract_features(img_path, xception_model)
img = Image.open(img_path)
description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
```

```
start a man in a red shirt and jeans is standing in front of a waterfall end
<matplotlib.image.AxesImage at 0x7e7982c70bb0>
```



Possible limitations:

- Image is not clear enough
- Figures are too far away
- Not enough similar image records

Actual Captions:

A boy dunks a basketball .	A boy dunks a basketball on a low hoop by the road .	A boy jumps up to make a basket in a basketball hoop .	A boy with a basketball , leaping for the goal .	The teenage boy is dunking a basketball in a hoop .

Bad Results

```
# Load the Xception model
xception_model = Xception(include_top=False, pooling="avg")

# Extract features from the image
photo = extract_features(img_path, xception_model)
img = Image.open(img_path)
description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
```

```
start a boy in a blue shirt is playing soccer end
<matplotlib.image.AxesImage at 0x7e7982add6c0>
```

Possible limitations:

- The model is trained too much on records with soccer

Analyze Results

- Given that our task is to generate a caption for an image, numerical metrics such as accuracy, precision, recall, F1-score, MAE, RMSE, etc., are irrelevant.
- To analyze the performance of the model, we can simply match how well our generated captions relate to the image.
- As shown in the previous slides, there are times when the caption generated is accurate and inaccurate.
    - The captions that are more accurate tend to be for more “common” images (such as images involving animals, sports, or single people)
    - The captions that are less accurate tend to be for “less common” images (such as images with multiple figures/people, many different colors, high contrast, etc.)
- The testing of the model is performed in a notebook called “test_model_8k.ipynb”, which is included in the email submission.

Moving to Flickr30k

- The model has room for improvement since it was only trained on 4800 images
- The next step is to train the model on the flickr30k dataset using 24811 images, if time permits
- Training the model on 24811 images is a huge task:
  - Extracting the features takes 6 hours
  - Training the model takes 22 hours

Training on Flickr30k (10 epochs)

```
2024-06-08 08:22:47.346767: I external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:998] successful NU
eturning NUMA node zero. See more at https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sy
2024-06-08 08:22:47.390547: W tensorflow/core/common_runtime/gpu/gpu_device.cc:2251] Cannot dlopen some GPU li
would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and set
Skipping registering GPU devices...
Model: "functional_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 78)	0	-
input_layer (InputLayer)	(None, 2048)	0	-
embedding (Embedding)	(None, 78, 256)	4,591,616	input_layer_1[0][0]
dropout (Dropout)	(None, 2048)	0	input_layer[0][0]
dropout_1 (Dropout)	(None, 78, 256)	0	embedding[0][0]
not_equal (NotEqual)	(None, 78)	0	input_layer_1[0][0]
dense (Dense)	(None, 256)	524,544	dropout[0][0]
lstm (LSTM)	(None, 256)	525,312	dropout_1[0][0], not_equal[0][0]
add (Add)	(None, 256)	0	dense[0][0], lstm[0][0]
dense_1 (Dense)	(None, 256)	65,792	add[0][0]
dense_2 (Dense)	(None, 17936)	4,609,552	dense_1[0][0]

```
 Total params: 10,316,816 (39.36 MB)
 Trainable params: 10,316,816 (39.36 MB)
 Non-trainable params: 0 (0.00 B)
None
24811/24811 ━━━━━━━━━━━━━━━━━━━━ 7806s 315ms/step - loss: 4.2760
24811/24811 ━━━━━━━━━━━━━━━━━━━━ 7781s 314ms/step - loss: 3.5237
24811/24811 ━━━━━━━━━━━━━━━━━━━━ 7768s 313ms/step - loss: 3.4198
24811/24811 ━━━━━━━━━━━━━━━━━━━━ 7807s 315ms/step - loss: 3.3791
24811/24811 ━━━━━━━━━━━━━━━━━━━━ 7819s 315ms/step - loss: 3.3606
24811/24811 ━━━━━━━━━━━━━━━━━━━━ 7769s 313ms/step - loss: 3.3538
24811/24811 ━━━━━━━━━━━━━━━━━━━━ 7798s 314ms/step - loss: 3.3503
24811/24811 ━━━━━━━━━━━━━━━━━━━━ 7857s 317ms/step - loss: 3.3502
24811/24811 ━━━━━━━━━━━━━━━━━━━━ 7883s 318ms/step - loss: 3.3516
24811/24811 ━━━━━━━━━━━━━━━━━━━━ 7827s 315ms/step - loss: 3.3545
(ML) ks3942@float:~/dsci471$ du -sh ~
3.6G    /home/ks3942
```

7800 seconds * 10 = 78000 seconds
78000 seconds = 1300 minutes
1300 minutes = **21.66 hours!**

Flickr30k Results

- Unfortunately, the results of the model for the Flickr30k dataset did not perform too well. The model performed around the same and a little worse than the model for Flickr8k, which is not what we expected since training on more images should theoretically yield better results.
- Because of how long it takes to train the model, we did not have enough time to tune any hyperparameters or improve the model's architecture.
- I trained the model 3 times trying to improve the model's performance (22 hours each time) but had no success.
- The testing of the model is performed in a notebook called "test_model_30k.ipynb", which is included in the email submission.

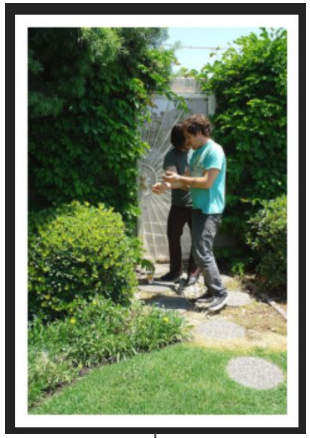
Transformer methods

```
learning_rate=0.01
weight_decay=0.0001
batch_size=128
num_epochs=1
patch_size=8
image_size=32
projection_dim= 256
num_patches = (image_size // patch_size) ** 2
epsilon=1e-6
dropout_rate = 0.1
attention_heads=1
```

Model: "vi_t"

Layer (type)	Output Shape	Param #
patch_generator (PatchGenerator)	(1, 16, 192)	0
patch_encoder (PatchEncoder)	(1, 16, 256)	53,504
transformer_encoder (TransformerEncoder)	(1, 16, 256)	527,104
layer_normalization_2 (LayerNormalization)	(1, 16, 256)	512
flatten (Flatten)	(1, 4096)	0
dropout (Dropout)	(1, 4096)	0
dense_3 (Dense)	(1, 512)	2,097,664
dense_4 (Dense)	?	5,130

Total params: 8,051,744 (30.71 MB)
Trainable params: 2,683,914 (10.24 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 5,367,830 (20.48 MB)

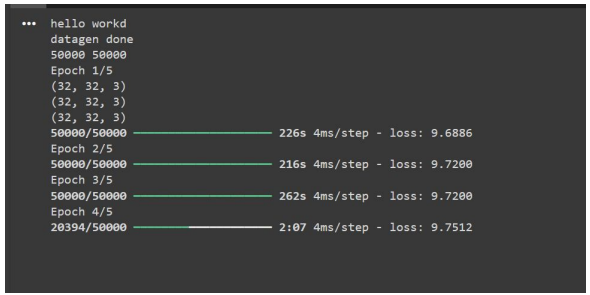


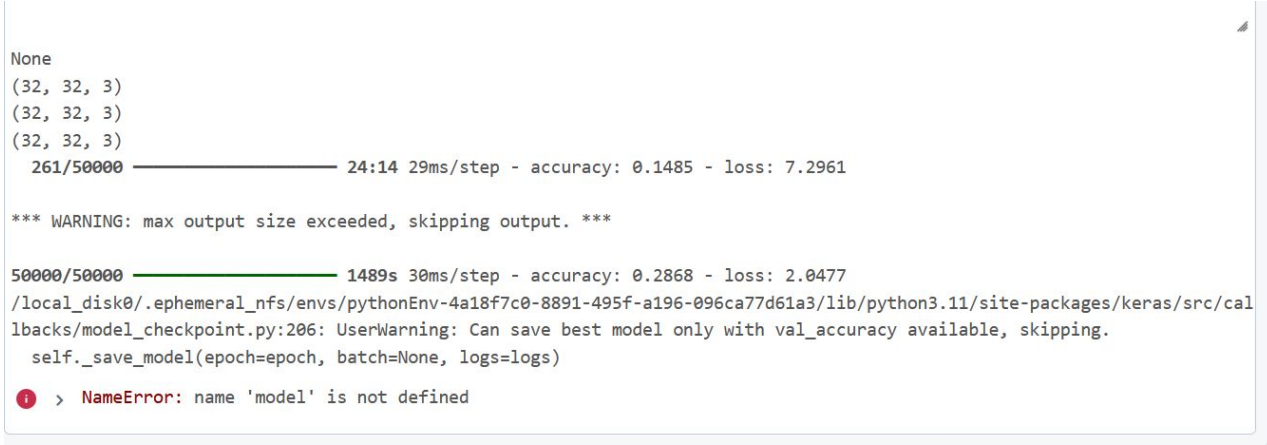
Problems with just ViT

- Hard to figure out which variation of the architecture is most applicable for our task
- Our initial goal was to design a vision transformer from scratch, to gain a greater understanding of transformers
- This meant we needed to train an image classifier from the ground up
- This worked badly enough to waste our time so we couldn't extensively try pre-trained models

```
drop_top(np.array(text_set['train'][1]['image'].resize((32,32))))

(32, 32, 3)
<tf.Tensor: shape=(1, 512), dtype=float32, numpy=
array([[-0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0., -0.,
        -0., -0., -0., -0., -0.]], dtype=float32)>

[ ] model(np.array(text_set['train'][3]['image'].resize((32,32))))

(32, 32, 3)
<tf.Tensor: shape=(1, 10), dtype=float32, numpy=
array([[-0.90833    , -0.8864399 , -0.89094186, -0.8104814 , -0.8240729 ,
        -1.188169   , -0.779306  , -1.0596313 , -0.97466964, -1.099249  ]],
      dtype=float32)>
```

Proof my transformer wasn’t wrong, just slow :(





Training the ViT on Databricks or even Colab with GPU allowed the model to reach upwards of 40% accuracy, but the model could not be saved so we are stuck with the model at 10% accuracy

Pretrained transformer with our LSTM

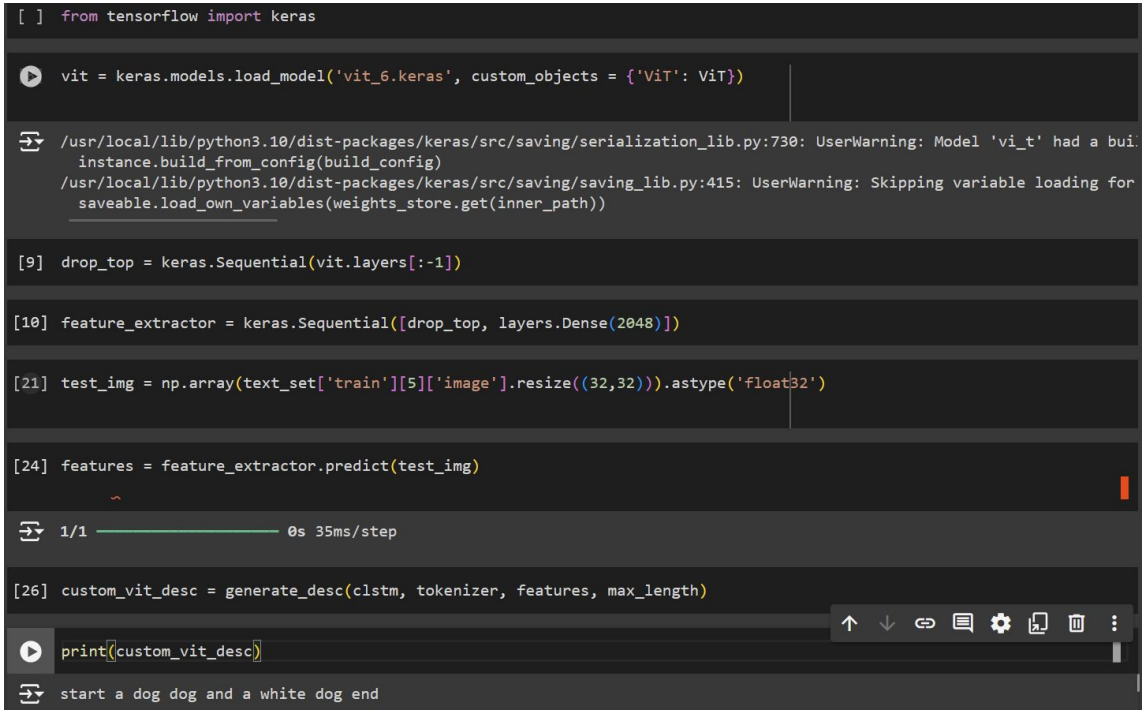
```
processor = AutoImageProcessor.from_pretrained("google/vit-base-patch16-224-in21k")
model = TFAutoModel.from_pretrained("google/vit-base-patch16-224-in21k")
```

```
inputs = processor(images=x_train[5], return_tensors = 'tf')

[19] lhs = model(inputs).pooler_output

[21] lhs.numpy()

(1, 768)

[32] densed = layers.Dense(2048)(lhs.numpy())
```

LSTM is trained on features produced by a CNN, so the hidden stats of a transformer are likely meaningless

```
[52] desc = generate_desc(clstm, tokenizer, densed, max_length)

[53] print(desc)

start a a a and a black and a dog in the a race end

x_train[5]

ndarray (369, 500, 3) show data
```

Custom ViT with our LSTM



```
[ ] from tensorflow import keras

vit = keras.models.load_model('vit_6.keras', custom_objects = {'ViT': ViT})

/usr/local/lib/python3.10/dist-packages/keras/src/saving/serialization_lib.py:730: UserWarning: Model 'vi_t' had a bui
  instance.build_from_config(build_config)
/usr/local/lib/python3.10/dist-packages/keras/src/saving/saving_lib.py:415: UserWarning: Skipping variable loading for
  saveable.load_own_variables(weights_store.get(inner_path))

[9] drop_top = keras.Sequential(vit.layers[:-1])

[10] feature_extractor = keras.Sequential([drop_top, layers.Dense(2048)])

[21] test_img = np.array(text_set['train'][5]['image'].resize((32,32))).astype('float32')

[24] features = feature_extractor.predict(test_img)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step

[26] custom_vit_desc = generate_desc(clstm, tokenizer, features, max_length)

print(custom_vit_desc)

start a dog dog and a white dog end
```

After initially presenting our project, we trained the ViT to 65% accuracy

This is the same image used to test the pre-trained transformer. A similar description has been generated, with the following tokens showing repeatedly:

- a
- dog
- black
- and

This indicates that our model is learning the same features as the pretrained model. Given more time, we can train the LSTM with features from our ViT, leading to better captions

Biggest Roadblock: Training

- This task is too large to run locally on our computers
- Databricks and colab have hard limits on how long you can run a notebook
  - Models take too long to train effectively, and saving the transformer model at various stages to train it across multiple sessions was not something we considered until it was too late
  - Even the CNN+LSTM posed issues with training times
- Tux has a hard limit of 4 GB on user storage
  - Model files were upwards of 1.5 GB
  - Installing tensorflow, keras, and datasets took over 2 GB
  - Model trained but failed to save the file due to storage limits
- Flickr8k dataset took around 4.44 hours to train
- Flickr30k dataset took around 22 hours to train

What We Learned

- CNN+LSTM can be used to create an image caption generator
  - CNN model called Xception is trained on imagenet dataset
  - Xception is responsible for image feature extraction
  - LSTM model uses the feature extraction to generate the image caption
- This was my (Kevin’s) first time using GPU on tux
  - Configuration/setting up virtual environment
  - Managing storage requirements
  - Navigating command line environment
- Challenges of transformer methods

Resources

- https://huggingface.co/datasets/nlphuji/flickr30k
- https://huggingface.co/datasets/jxie/flickr8k/viewer/default/train?p=1&row=147
- https://www.analyticsvidhya.com/blog/2021/12/step-by-step-guide-to-build-image-caption-generator-using-deep-learning/
- https://data-flair.training/blogs/python-based-project-image-caption-generator-cnn/